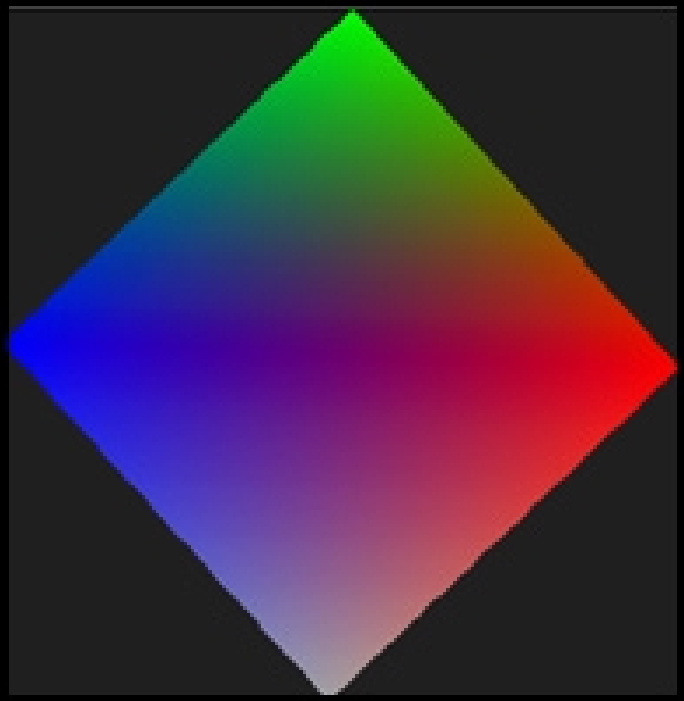
Session 405

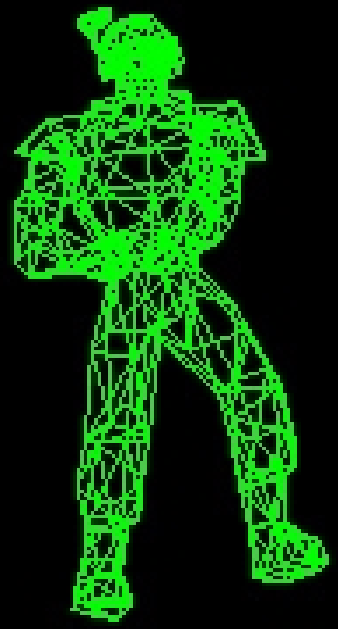
OpenGL:
Geometry and Modeling

Todd Previte
3D Graphics DTS Engineer

Purpose

- Implementation of Complex Animated Geometries





What You'll Learn

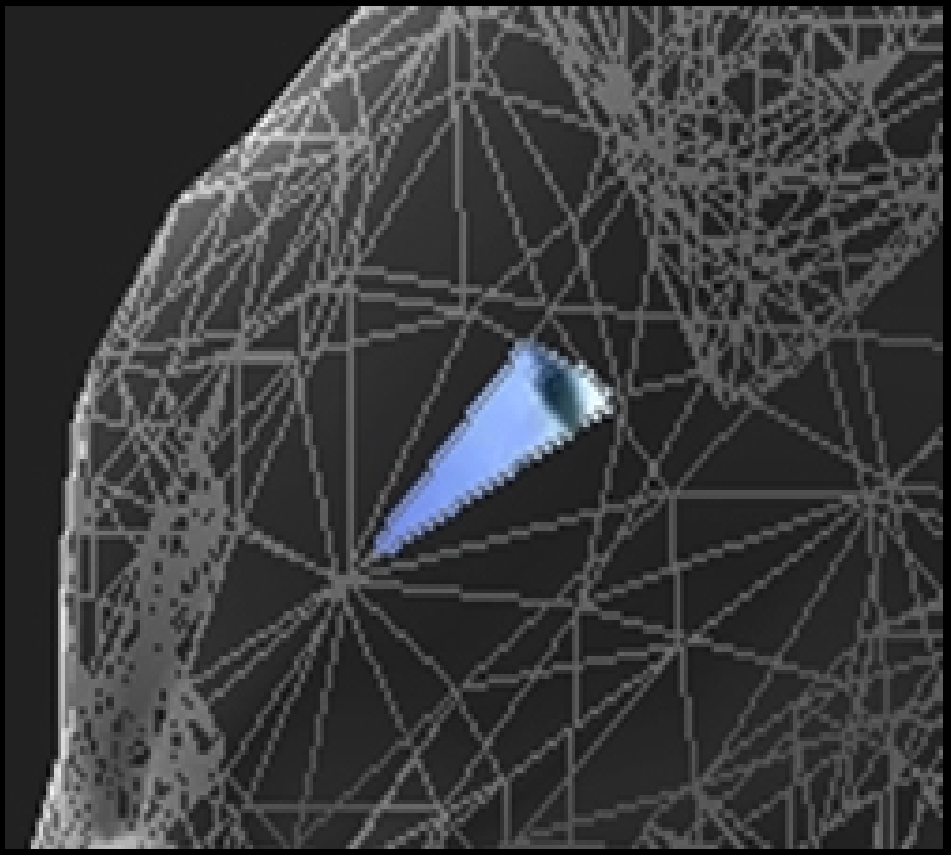
- Model formats
- How to
  - Load from a file
  - Store internally
  - Display
  - Animate

Coordinate Systems

- Texture coordinates
  - Normally 2D
- Model (or object) coordinates
  - Local to model
- World coordinates
  - Global reference
- Eye coordinates
  - Resulting coordinates from the modelview matrix transform

Coordinate Systems

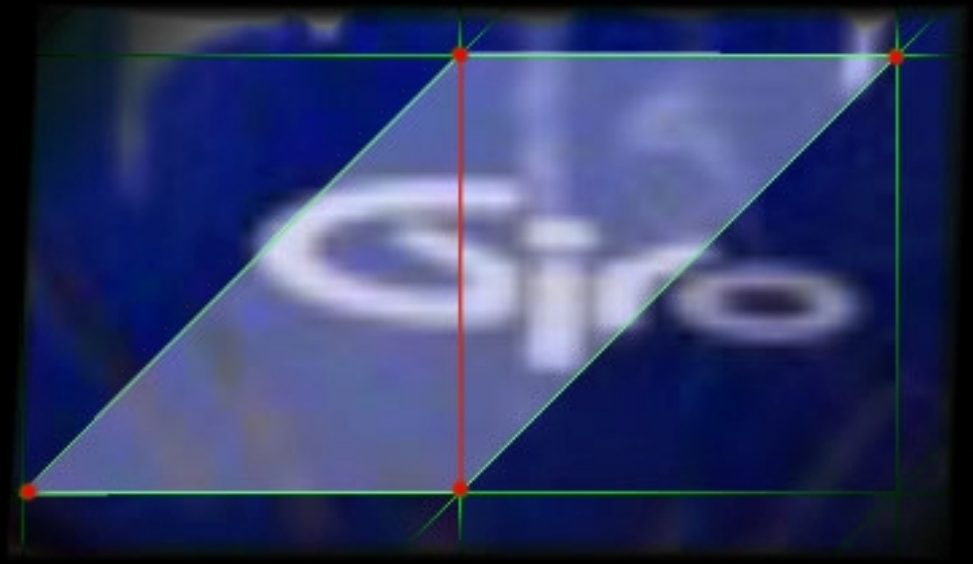
- Coordinate Systems



Model Formats

- What's In A Model Format
  - Vertices
    - Array of 3 floats(x,y,z) in model space
  - Texture coordinates
    - Array of 2 floats(s,t) in texture image space
    - Not always equal to number of vertices
  - Triangles
    - Array of 3 vertex and texture coordinate indices
  - Textures
    - Image data

Model Formats Sidebar



- Per vertex—Same texture coordinate will cause problems if each triangle uses its own texture
- Per triangle vertex—shared vertices have one texture coordinate per triangle that shares that vertex

Model Formats

- What's In A Model Format
  - Vertices
    - Array of 3 floats(x,y,z) in model space
  - Texture coordinates
    - Array of 2 floats(s,t) in texture image space
    - Not always equal to number of vertices
  - Triangles
    - Array of 3 vertex and texture coordinate indices
  - Textures
    - Image data

Model Format Case Study

- Quake 2 (.md2)
  - Publicly available
  - Easy to understand
  - Well documented
  - Many models readily available

Quake 2 (.md2) Format

- .MD2 File Header

```
typedef struct
{
   int magic;                    int numGlCommands;
   int version;                  int numFrames;
   int skinWidth;                int offsetSkins;
   int skinHeight;               int offsetTexCoords;
   int frameSize;                int offsetTriangles;
   int numSkins;                 int offsetFrames;
   int numVertices;              int offsetGlCommands;
   int numTexCoords;             int offsetEnd;
   int numTriangles;
} model_t;
```

Quake 2 (.md2) Format

- Data Representation
  - Vertex Data

```
triangleVertex_t vtxArray[numVertices];
        typedef struct  {
                byte vertex [3];
                byte lightNormalIndex;
                } triangleVertex_t;
```

  - Triangle Data

```
triangle_t triArray[numTriangles];
        typedef struct  {
                byte vertexIndices [3];
                byte textureIndices [3];
                } triangle_t;
```

Display

- Encoded byte coordinate
  - 8 bits multiplied by ‘scale’ plus ‘translate’
    - Yields a floating point number that can be used by OpenGL as a coordinate
    - Conserves memory
    - Makes loading easier
    - Allows for easy scaling of objects as well as moving the objects around

Quake 2 (.md2) Format

- Data Representation
  - Texture Data
    - Texture names
      - Stored as a char[64] in the file
      - Image size will always be **skinWidth** and **skinHeight** for any given .md2
    - Texture Coords

```
textureCoordinate_t texCoords[numTexCoords];
typedef struct  {
        short s;
        short t;
        } textureCoordinate_t;
```

Quake 2 (.md2) Format

- Data Representation
  - Frame Data

```
frame_t frames[numFrames];
    typedef struct {
        float scale[3];
        float translate[3];
        char name[16];
        triangleVertex_t vertices[1];
        } frame_t;
```

Storage

- Internal Storage Of A .md2 Model

```
typedef struct {
        model_t md2Header;
        frame_t *frames;
        triangle_t *triangles;
        textureCoordinate_t *texCoords;
} Q2Model;
```

Loading

- File loading from a .md2 file

```
//Find and open model file
...
// Read in the header
fread (modelFile, sizeof (model_t), &model.md2Header);
// Now read in the other data using the offsets from the header
fseek (fileBegin + model.ms2Header.offsetFrames);
model.frames = (frame_t*)
        NewPtrClear (model.md2Header.frameSize *
                model.md2Header.numFrames);
fread (modelFile,
   model.md2Header.frameSize * model.md2Header.numFrames,
   &model.frames);
```

Loading

- File loading from a .md2 file

```
//Find and open model file
...
// Read in the header
fread (modelFile, sizeof (model_t), &model.md2Header);
// Now read in the other data using the offsets from the header
fseek (fileBegin + model.ms2Header.offsetFrames);
model.frames = (frame_t*)
        NewPtrClear (model.md2Header.frameSize *
                model.md2Header.numFrames);
fread (modelFile,
   model.md2Header.frameSize * model.md2Header.numFrames,
   &model.frames);
```

Loading

- File loading from a .md2 file

```
//Find and open model file
...
// Read in the header
fread (modelFile, sizeof (model_t), &model.md2Header);
// Now read in the other data using the offsets from the header
fseek (fileBegin + model.ms2Header.offsetFrames);
model.frames = (frame_t*)
        NewPtrClear (model.md2Header.frameSize *
                model.md2Header.numFrames);
fread (modelFile,
   model.md2Header.frameSize * model.md2Header.numFrames,
   &model.frames);
```

Loading

- File loading from a .md2 file

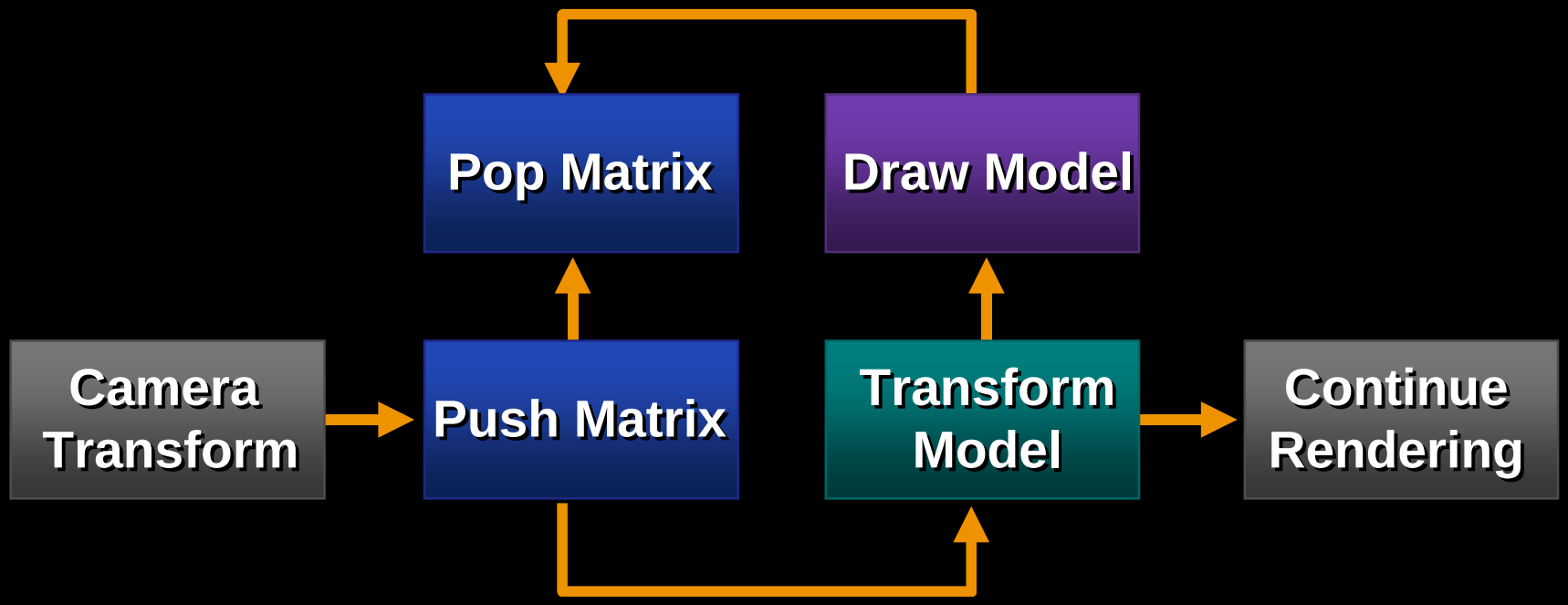
```
//Find and open model file
...
// Read in the header
fread (modelFile, sizeof (model_t), &model.md2Header);
// Now read in the other data using the offsets from the header
fseek (fileBegin + model.ms2Header.offsetFrames);
model.frames = (frame_t*)
        NewPtrClear (model.md2Header.frameSize *
                model.md2Header.numFrames);
fread (modelFile,
   model.md2Header.frameSize * model.md2Header.numFrames,
   &model.frames);
```

Display

- Drawing models
  - Outline for rendering models in a draw loop
    - Transform camera
    - Push matrix
    - Transform model
    - Draw
    - Pop matrix (back to camera transform)
    - Repeat for next model

Display

- Drawing models
  - Graphical representation of a drawing loop



Display

- Drawing models

```
void RenderScene ( Q2Model* modelList, long modelCount) {
        // Camera setup and transforms
        // Draw stuff here such as backgrounds, skies, etc.
        for ( int q = 0; q < modelCount; q++) {
                glPushMatrix();
                // Model transforms here, then draw it
                DrawModel( &modelList [ q ], frameNum);
                glPopMatrix();
        }
        // Other rendering stuff
}
```

Display

- Model Rendering Routine

```
void DrawModel (Q2Model* theModel, int k) {
        int vi [ 3 ];  float vtx [ 3 ];
        glBegin(GL_TRIANGLES);
        for (int i = 0  ; i < theModel.md2Header.numTriangles; i++) {
          for (int j = 0 ; j < 3 ; j++) {
                vi [ j ] = theModel.triangles [ i ].vertexIndicies [ j ];
                for ( int m = 0 ; m < 3 ; m++) {
                  vtx [ m ] = frame [ k ].vertices [ vi [ j ] ].vertex [ m ] *
                              frame [ k ].scale [ m ] +
                              frame [ k ].translate [ m ];
                glVertex3d(vtx [ 0 ], vtx [ 1 ], vtx [ 2 ]); }
} } glEnd(); }
```

Display

- Model Rendering Routine

```
void DrawModel (Q2Model* theModel, int k) {
        int vi [ 3 ];  float vtx [ 3 ];
        glBegin(GL_TRIANGLES);
        for (int i = 0  ; i < theModel.md2Header.numTriangles; i++) {
          for (int j = 0 ; j < 3 ; j++) {
                vi [ j ] = theModel.triangles [ i ].vertexIndicies [ j ];
                for ( int m = 0 ; m < 3 ; m++) {
                  vtx [ m ] = frame [ k ].vertices [ vi [ j ] ].vertex [ m ] *
                              frame [ k ].scale [ m ] +
                              frame [ k ].translate [ m ];
                glVertex3d(vtx [ 0 ], vtx [ 1 ], vtx [ 2 ]); }
} } glEnd(); }
```

Display

- Model Rendering Routine

```
void DrawModel (Q2Model* theModel, int k) {
        int vi [ 3 ];  float vtx [ 3 ];
        glBegin(GL_TRIANGLES);
        for (int i = 0  ; i < theModel.md2Header.numTriangles; i++) {
          for (int j = 0 ; j < 3 ; j++) {
                vi [ j ] = theModel.triangles [ i ].vertexIndicies [ j ];
                for ( int m = 0 ; m < 3 ; m++) {
                  vtx [ m ] = frame [ k ].vertices [ vi [ j ] ].vertex [ m ] *
                              frame [ k ].scale [ m ] +
                              frame [ k ].translate [ m ];
                glVertex3d(vtx [ 0 ], vtx [ 1 ], vtx [ 2 ]); }
} } glEnd(); }
```

Display

- Model Rendering Routine

```
void DrawModel (Q2Model* theModel, int k) {
        int vi [ 3 ];  float vtx [ 3 ];
        glBegin(GL_TRIANGLES);
        for (int i = 0  ; i < theModel.md2Header.numTriangles; i++) {
          for (int j = 0 ; j < 3 ; j++) {
                vi [ j ] = theModel.triangles [ i ].vertexIndicies [ j ];
                for ( int m = 0 ; m < 3 ; m++) {
                  vtx [ m ] = frame [ k ].vertices [ vi [ j ] ].vertex [ m ] *
                              frame [ k ].scale [ m ] +
                              frame [ k ].translate [ m ];
                glVertex3d(vtx [ 0 ], vtx [ 1 ], vtx [ 2 ]); }
} } glEnd(); }
```

Display

- Drawing Models
  - Simple model rendering routine
    - Omits animations
    - Only renders 1 particular frame
    - No textures
  - For each vertex coordinate:
    - Each vertex is an encoded byte
    - Multiply by frame scale
    - Add frame translation

Animation

- Key frame based
- Vertex interpolation between key frames

Animation

- Key Frame Animation
  - Each frame contains a complete set of vertex data for the animated model
  - Series of “snapshots” over time
  - Simple rigid body motion

Animation

- Key Frame Animation—Graphical Representation

Animation

- Vertex Interpolation Between Frames
  - Vertices are interpolated linearly
  - Number of interpolated frames is arbitrary
    - Based on frame rate and animation speed
    - # of Interpolations = Animation speed (frame to frame) / Time to render 1 frame
    - 0.5 second animation speed (500ms) / 30 frames per second (~33ms) = 15.0 interpolated frames

DEMO: Model Animation

Summary

- Model Formats
  - Definitions
  - Necessary data
- Loading a model file
- Internal storage and representation of model data
- Display a model
- Animating a model

For More Information

- OpenGL
  - **http://www.apple.com/opengl**
  - **http://www.opengl.org**
  - *OpenGL Programming Guide*, *OpenGL ARB*, Addison-Westley
- Modeling
  - *Real-Time Rendering*, Moller | Haines, A K Peters, Ltd.
  - *3D Graphics File Formats* , Keith Rule, Addison-Westley
- Animation
  - *Principles of Three-Dimensional Computer Animation*, Michael O'Rourke-Norton
  - *Advanced Animation and Rendering Techniques*, Watt | Watt, Addison-Westley

Roadmap

408 OpenGL: Optimization	Hall J2 **Thurs., 9:00am**
409 OpenGL: Advanced Rendering	Hall J2 **Thurs., 10:30am**
FF10 Feedback Forum: OpenGL	Hall J1 **Thurs., 5:00pm**

Who to Contact

Todd Previte
3D Graphics DTS Engineer
Apple Worldwide Developer Relations
tprevite@apple.com

http://developer.apple.com/events/urls.html

Q&A

Todd Previte
3D Graphics DTS Engineer
Apple Worldwide Developer Relations
tprevite@apple.com

ANNOUNCING

QuickTime Live!

October 8–11, 2001
The Beverly Hilton Hotel
Beverly Hills, CA

Full day workshops, Conference Sessions,
QuickTime Product Showcase, Keynotes and Social Events with the community
of QuickTime Developers, Content Creators, and New Media Business Professionals.
Get Face-to-Face with the QuickTime team for a week of training, business
Opportunities and industry perspectives that you can't get anywhere else.

www.apple.com/quicktimelive

Develop. Produce. Deliver.

WWDC

Worldwide Developers Conference 2001